Queue-Pair IP: A Hybrid Architecture for System Area Networks

Phil Buonadonna, and David Culler

IRB-TR-02-002

March, 2002

Intel Research Berkeley

Abstract

We propose a SAN architecture called Queue Pair IP (QPIP) that combines the interface from industry proposals for low overhead, high bandwidth networks, e.g. Infiniband, with the well established inter-network protocol suite. We evaluate how effectively the queue pair abstraction enables inter-network protocol offload. We develop a prototype QPIP system that implements basic queue pair operations over a subset of TCP, UDP and IPv6 protocols using a programmable network adapter,. We assess this prototype in terms of basic application performance, underlying processing costs, and a network storage application. With modest hardware support, QPIP can perform as well as traditional inter-network protocol implementations at a fraction of the host CPU overhead. With hardware support equivalent to Infiniband, QPIP would achieve similar performance targets.

1. Introduction

I/O systems today are on the verge of an architectural revolution motivated by ever-increasing interconnectivity demands. Present system architectures form a bottleneck that limits performance. Processing units communicate with I/O devices using low latency load/store semantics or DMA across a shared bandwidth bus, such as the Peripheral Component Interconnect (PCI). The bus extends at most a few feet from the processor, and supports a limited number of devices and minimal fault-tolerance.

A proposed solution to this problem is to merge present I/O architectures and modern networks into a new paradigm of I/O based networks. Processor/Memory combinations and I/O devices (*e.g.* disks and WAN adapters) are connected directly to a switched interconnect fabric and communicate through network-oriented protocols. Communication can be processor-to-processor or processor-to-device. The switch-based design permits a large array of devices to be connected in a manner that provides scalable throughput and the network protocols provide for a high degree of fault-tolerance. However, network latency must be comparable to the µsec latency of busses, with few instructions of host overhead. The class of networks that support this concept of networked I/O have been termed System Area Networks (SAN).

A key component to the success of SANs is developing a standardized architecture around which interoperable systems can be designed and built. At present, there are two prominent, and somewhat competing, efforts toward this goal. One of these is a cooperative industry standard called Infiniband [17]. At the other is a series of IETF groups looking to employ established inter-network protocols[1] in the SAN environment.

Infiniband is the merger of several industry projects in SAN architectures: the VI Architecture, Next Generation I/O and Future I/O. The core concept of Infiniband is to interconnect host processor/memory modules and I/O devices by a switched network fabric, effectively eliminating the traditional I/O bus. In order to meet performance targets of low latency, high bandwidth and low host overhead, Infiniband proposes extensive hardware support combined with a completely new communication stack. At the top, it defines a new communication abstraction called the Queue Pair (QP). It also specifies the management transport, network, link, and physical layers.

By comparison, Inter-network protocol proposals in this area define only the higher-layers of the network stack. There are separate specs for the management, transport and network layers, and there are no specific assumptions about the underlying link or physical layers. Historically, traditional operating system interfaces to simple network adapters have been successful. Additionally, as processors and links have become faster, these implementations have demonstrated increased bandwidth. However, the host-based nature of these implementations has caused debate on whether or not host overhead and latency can meet the performance demands of the SAN. Efforts to accelerate inter-network protocol stacks have met with limited success as discussed below. The majority of these sought to maintain the traditional communication abstraction (*i.e.* sockets) and host-based nature of the stack, while optimizing implementation internals either though software techniques or link layer assistance.

In this paper, we propose a SAN architecture called Queue Pair IP (QPIP) that combines modern industry proposals for high-performance networks with the well established inter-network protocol suite. Specifically, the

[1] This is paper, *inter-network protocols* refers to the suite of established IETF protocols that include TCP, UDP and IP. It can be used interchangeably with *IP protocols.*

architecture implements the Queue Pair (QP) communication abstraction proposed by industry efforts directly over standard inter-network protocols implemented in an intelligent network interface.

Our hypothesis is that the queue pair abstraction enables effective offload of inter-network protocols onto the network interface. Using a programmable network adapter, we have developed a prototype QPIP system that implements basic QP operations over a subset of TCP, UDP and IPv6 protocols. We assess this prototype in terms of basic performance, host overhead, network interface occupancy and network storage performance. Our results show that QPIP can perform equal to or better than traditional inter-network protocol implementations at a fraction of the host CPU overhead. Further, the QPIP model enables a straightforward design that does not require highly specialized or complex hardware to support this level of performance. Such simplification provides an opportunity for specialized hardware acceleration that would enable meeting the performance requirements of the SAN regime in an interoperable framework.

The rest of this paper is organized as follows. Section 2 presents a background of work in System Area Networks, Infiniband, and inter-network protocols. Section 3 describes the QPIP architecture. Section 4 details our QPIP prototype and presents an analysis of its performance. Section 5 discusses key lessons learned from the QPIP implementation. Section 6 presents our conclusions and future work.

2. Background and Related Work

2.1. System Area Networks & Infiniband

There have been several efforts aimed at high performance interfaces for system area networks [12, 23, 27, 30, 31, 33, 34]. These have explored a variety of concepts including usage abstractions and OS bypass mechanisms. The results demonstrate that changing the usage abstraction enables low overhead, high bandwidth communication.

Infiniband draws upon these previous research efforts and utilizes a memory-based user-level communication abstraction. The communication interface in Infiniband is the *Queue Pair* (QP), which is the logical endpoint of a communication link. The QP is a memory-based abstraction where communication is achieved through direct memory-to-memory transfers between applications and devices. It consists of a send and a receive queue of work requests (WR). Each WR contains the necessary meta-data for the message transaction including pointers into registered buffers to receive/transmit data to/from.

There are two classes of message transactions in the QP model: *send-receive* and *remote DMA* (RDMA). To conduct transfers, the user process constructs a WR and posts it to a QP. The posting method adds the WR to the appropriate queue and notifies the adapter of a pending operation. In the send-receive paradigm, the target pre-posts receive WRs that identify memory regions where incoming data will be placed. The source posts a send WR that identifies the data to send. Each send operation on the source consumes a receive WR on the target. In this scheme, each application manages its own buffer space and neither end of the message transaction has explicit information about the peer's registered buffers. In contrast, RDMA messages identify both the source and destination buffers. Data can be directly written to or read from a remote address space without involving the target process. However, both processes must exchange information regarding their registered buffers using some out-of-band mechanism such as a send-receive operation.

Closely coupled with the QP is the *completion queue* (CQ). When a QP is created, its individual send and receive channels are bound to one or separate CQs. A single CQ may have multiple QPs associated with it. When a WR completes, a token is added to the completion queue and can be detected by the application through polling or an event. The binding of multiple queues to a CQ permits applications to group related QPs into a single monitoring point.

The QP and CQ functionality is implemented in the Infiniband network interface or *channel adapter*. Channel adapters incorporate a high degree of hardware support to minimize host overhead and protocol processing occupancy in the interface itself.

Infiniband also defines the mechanisms beneath the communication abstraction. It includes specifications for the transport reliability levels, connection and connectionless communication, management, network routing, link layer formats and physical layer properties. With the exception of using the IPv6 addressing scheme for inter-subnet packets, it is largely divested from standard inter-network protocols. The architecture does export a raw link layer interface that can be used by traditional inter-network protocol implementations, but it does not provide the same performance guarantees. The QPIP architecture seeks to invert this strategy and make inter-network protocols the core transport underneath the QP model.

2.2. Inter-network Protocols

Standard inter-network protocols have seen a great deal of success in usage and throughput capacity. Their high-level, link-independent nature has permitted deployment on a wide variety of platforms and enjoyed a tremendous growth in bandwidth. Additionally, the open forum of development has permitted continued refinement and introduction of new application spaces. However, the general nature of these protocols has led to principally host-based software implementations that incur non-negligible overhead on the host processors that impact latency and other computation. Thus, it is frequently perceived that inter-network protocols are not suited for SAN environments.

Detailed analysis of inter-network protocols [3, 19-22, 28] have identified host overhead and bottlenecks for the TCP/IP protocol combination. There have been efforts to mitigate these problems through techniques such as Integrated Layer processing [1, 5], reducing caches misses and memory cycles [4, 25] and *fbufs* [10]. The thrust of all these efforts has been to focus on internal protocol mechanisms and buffer management without changing the application interface. Even so, significant host overhead remains. QPIP advocates an alternative memory-based interface that eliminates major sources of overhead and allows the complete offload and a simplified implementation of inter-network protocols.

Afterburner [8] explored providing hardware support for minimizing data movement in kernel based inter-network protocol implementations. Edwards [13, 14] used Afterburner in conjunction with Jetstream [35] for user-space inter-network protocol implementations. The core trait of these projects was to optimize mechanisms at the lower-level, thus only eliminating some of the overhead. The bulk of the inter-network stack remained on the host and used a traditional interface.

The OSIRIS network adapter [9, 11], U-Net [33], Arsenic [29], Trapeze [16], and Windows Sockets Direct Path [24] explored techniques for direct user-level access to network hardware with an inter-network protocol stack as an application library. This demonstrated some performance improvement, but the inter-network communication path was still significantly slower than the direct access path to the hardware and incurred host overhead associated with the user-level protocol library. QPIP seeks to make the inter-network stack *the* fast path and provide a light weight abstraction on top instead of underneath the stack.

Pushing inter-network protocol processing all the way to the interface has also been examined. However, many of the historical efforts are not well documented. The Nectar communications processor provided fully offloaded network functionality, including TCP/IP, under a user-level interface based on shared memory semantics [7]. Using this technique, Nectar demonstrated TCP throughput of 24 Mbit/sec vs. 6.4 Mbits/sec when employed as a simple link layer underneath a host-based stack. However, best performance was achieved by using a specialized protocol on the adapter. Moreover, the TCP/IP stack in Nectar was used primarily to show the flexibility of the complex runtime system. The complexity of the Nectar design led to a lengthy development cycle and prevented it from keeping pace with host-based stacks on faster host processors. QPIP builds on the Nectar approach by making the inter-network stack a core component of the runtime system. It employs a simple interface that can be implemented either on programmable NIs on a competitive growth curve or by specialized hardware design.

Recent industry efforts have sought to utilize special hardware to achieve inter-network protocol acceleration. Alacritech's SLIC [2] provides support for common TCP/IP data movement, including zero-copy, underneath existing socket abstractions. QPIP explores full protocol offload for all protocol functions underneath a lightweight memory based abstraction. The iReady Internet Tuner [18] provides a full inter-network protocol implementation in a chip, but is designed for embedded device applications using a serial interface. It only supports a small number of connections and does not have a high performance interface. The Emulex GN9000 [15] supports a VI Architecture abstraction over TCP/IP protocols, but adds an additional protocol layer between TCP and the application interface.

QPIP seeks to implement the QP abstraction directly on top of the TCP/UDP transport layer without an intervening protocol layer.

3. The QPIP Architecture

The QPIP architecture is a hybrid approach that combines the QP communication model over inter-network protocols implemented in an intelligent adapter. Using inter-network protocols eliminates the need to re-engineer the transport, network and link layer components and brings the wealth of understanding and services of these protocols to the SAN space.

At the top of the network stack, the QP replaces the socket as the communication abstraction. Communication is achieved by sending and receiving messages on a QP as opposed to a series of `read()` and `write()` calls to a socket. Beneath this, the transport (UDP/TCP), network (IP) and link layers are processed within the network interface. These can be simplified because of the application interface and the intimacy with the network.

A key design point of the QPIP architecture is that it uses established protocol formats for inter-network protocols and does not add any additional protocol formats. This provides a straightforward means to bridge the SAN to external networks using conventional means such as a firewall or router. Communication can occur between QPIP applications or QPIP and traditional (socket) systems. QP to QP is the high performance mode for communication within the SAN. In the latter mode, the remote end sees a conventional IP sockets, but the QP end is aware of the remote limitations and may have to re-assemble incoming data into a complete unit. This reassembly could be done by an optional library or perhaps with assistance from the interface.

For best effort datagrams using UDP, a QP is created that is bound to a particular UDP port. The WRs in a UDP QP identify the target or source address/port for sent or received messages respectively. Data is encapsulated directly in the UDP datagrams without an additional protocol layer. As soon as a UDP message is sent, the associated send WR is marked as complete. Incoming UDP datagrams, from either another QP or a socket, consume a WR in the destination QP.

For reliable communication using TCP, a connection is established between two QPs using the same rendezvous model as sockets, avoiding the need to define a new connect protocol (sec. 12 of [17]). The server application instructs the interface to monitor a TCP port for incoming connections. The client creates a QP, binds it to a TCP port, and initiates a connection to the server that mates the connection to an idle QP in the server application. Connection establishment uses standard SYN-ACK state processing of TCP. However, the handshake is handled in the interface with the host only being notified when the connection is established. When a send WR is posted, the message data is encapsulated into TCP segments for delivery to the target QP. This WR completes when all the data for that message is acknowledged by the destination.

Identifying message boundaries in the TCP byte stream for QP-to-QP communication can be accomplished in a variety of ways and we explore one method in section 4. For QP-to-socket connections, there is no framing information available, thus the application may have to reassemble individual segments into a complete message.

The CQ mechanism in QPIP performs the same functionality as the `select()` system call for traditional interfaces. However, the CQ is the primary mechanism for detecting completions and is not optional.

3.1. Organization

The QPIP network interface provides the core functionality of the QPIP architecture. Its operation can be broken down into four logical finite state machines (FSMs) as shown in Figure 1. A common data structure is used to maintain the state of the individual QPs and includes the inter-network protocol specific information, namely the TCP transmission control block (TCB). The doorbell FSM, a key concept from the VI Architecture and implicit in Infiniband, continuously monitors the QP notification mechanisms and updates the state table with the number of outstanding WRs in the queue. The management FSM processes privileged commands from the host operating system. These include QP/CQ creation and removal, establishment of registered memory bindings and QP connection management.

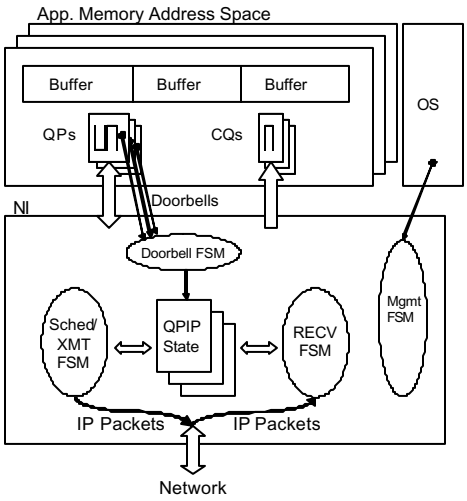

Figure 2. The QPIP organization.

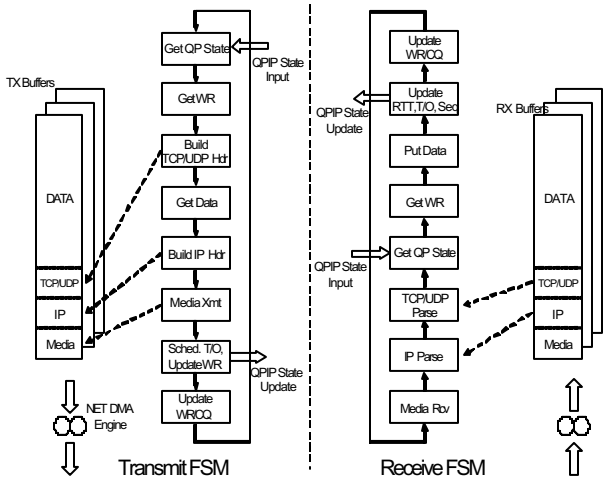

Figure 1. QPIP transmit and receive state machines and data structures.

The other two state machines form the communication core, as illustrated in detail in Figure 2. The transmit finite state machine serves as the scheduler and transmit control path. The state machine polls all active communication endpoints for outstanding data that needs to be sent. In the case of TCP, it additionally monitors for timeout/retransmit events pending on a QP. If a transmit or retransmit operation needs to occur, the state machine services the endpoint. It obtains and parses the appropriate work request from the QP, identifies the data to send, and then builds the necessary TCP/UDP and IP headers. Once a packet is formatted, it is encapsulated in a link layer header and pushed into the network. For TCP, the state machine sets the necessary timeout events and updates the TCB state in the QPIP state table. Finally, any status updates are posted back to the WR and the associated QP. Unless there is an error, TCP based communication will not post completion updates at this stage, as this will be done when a packet is acknowledged.

The receive state machine operates in the reverse fashion. It is invoked whenever a packet arrives from the network. Of note, a pure TCP acknowledgement is simply a special case of a regular data receive operation, except that no data is delivered to the application. Instead, the WR that *sent* the data being acknowledged is updated with a completion status and an appropriate token entered into its completion queue.

4. QPIP Prototype

We have implemented a functioning QPIP prototype on a PCI based system with a widely used SAN. On this prototype, we have conducted basic performance benchmarks and compared them to existing network implementations. We also evaluated performance in the context of the Network Block Device storage application.

4.1. Design

The prototype uses the Myrinet SAN with the LANai 9 programmable network interface as the QPIP intelligent NIC. The interface has a 133 MHz general purpose RISC processor, 2 megabytes of on-board SRAM, two PCI DMA engines and 2 network transfer engines (send and receive). The network itself is switched and uses source-based, oblivious cut-through routing. Individual links operate at 2.0 Gbs (full-duplex) and support arbitrary sized MTUs. The DMA controller hardware includes support for computing IP checksums and has a specialized doorbell mechanism where writes to a region of PCI address space are stored in a FIFO in the interface SRAM. The programmable nature of this interface permitted quick prototyping and could be instrumented to provide performance details.

The implementation consists of three major software components: a kernel driver, an application software library and the network interface firmware. The kernel driver provides basic hardware initialization and mechanisms for mapping address space and translating virtual to physical addresses for registered buffers. It also provides a lightweight interrupt service routine to process events. The library provides the basic communication methods -

`PostSend()`, `PostRecv()`, `Poll()` and `Wait()` - as well as communication management functions. Internal details of the QP and CQ structures are hidden from the application by the library. These structures are resident in host memory and are accessed by the NIC through DMA. The library methods can be used both in the kernel and directly at user level. The network interface firmware provides the core functionality of the QPIP architecture. It implements the four state machines and data structures as outlined in the previous section and provides a functional subset of the QP operations and inter-network protocols. It supports the send-receive class of QP messaging in both unreliable (UDP) and reliable (TCP) delivery modes. It also includes a facility for translating virtual addresses in WRs to physical addresses for use in DMA transactions.

Underneath the QP abstraction, the firmware implements a subset of the TCP, UDP and IPv6 standard protocols. The implementations are based on existing inter-network protocol stacks to shorten development time and ensure correctness. The protocol subsets are sufficient to provide meaningful comparisons with existing implementations under common-case network conditions in the SAN environment. Specifically, we assume the network to be robust and packet loss or reordering seldom occurs. The common-cases are specifically those that meet the header prediction requirements as presented in [32].

The IPv6 protocol layer is based on the FreeBSD v4.x distribution and provides support for basic connectivity. Address resolution is provided by a static table that maps IPv6 addresses to switch routes. We chose the IPv6 protocol as we believe it reflects the next generation of network systems. Although we don't explore fragmentation issues, the IPv6 standard supports only end-to-end fragmentation which is better suited to hardware based protocol implementations.

The UDP and TCP implementations are based on those found in [6, 32]. The UDP protocol is fully functional. Unreliable QP messages are encapsulated directly in UDP datagrams for transmission over the network. The TCP stack implements RTT estimation, window management, and congestion and flow control mechanisms. It also includes RFC 1323 time-stamp and window scaling performance enhancements. Support for out-of-order reassembly or urgent data was not included. For this prototype, we chose to map QP messages one-for-one onto TCP segments (i.e. a segment *is* a message). This mapping is fairly straightforward to implement and does not require additional information to be included in the packets. The tradeoff being that the TCP segments are arbitrarily sized and performance could suffer if subsequent IP fragments are lost. However, in the SAN environment, loss is expected to be minimal and the benefits of this simple mapping can be exploited.

4.2. Performance

Analysis of the QPIP prototype was conducted using Dell Poweredge 6350 servers (4x550MHz Pentium-III processors, 1GB of physical RAM and a 64-bit/33MHz PCI bus) running Linux version 2.4. Installed on the I/O bus were the Myrinet LANai 9 NIC and an Intel Pro1000 Gigabit Server Adapter.

4.2.1. Basic communication benchmarks. The basic communication benchmarks included application to application round-trip time (RTT) and throughput. The round-trip time refers to the latency of a single 1 byte message to travel from one application to another and back. Throughput is the maximum sustainable bandwidth between two applications. We compare the QPIP results with the Linux host-based IPv4 stack running over Gigabit Ethernet (1500 Byte MTU) and the Myrinet adapter running Myricom's GM v.1.4 software (9000 Byte MTU). Due to an artifact of the Myrinet hardware, the QPIP prototype cannot exploit hardware assisted IP checksums on the receive side. As we are interested in a high-level comparison of the architecture, the results in the figures were obtained using an emulated hardware IP checksum on the receive-side. For completeness, we also report the results obtained using firmware based checksumming.

Figure 3 presents the RTT comparison for the three implementations for both TCP and UDP protocols. Using a firmware checksum, the QPIP latency is 73µsec (UDP) and 113 µsec (TCP). We perform a closer analysis of the RTT later in this section.

Throughput results were derived from the *ttcp* (v1.4) benchmark. The tests involved a 10MB transfer in 16KB chunks with the TCP_NODELAY option set. Figure 4 presents the throughput and CPU utilization results for the three implementations using native MTUs (16KB in the case of QPIP). Using a firmware based checksum on the QPIP prototype, the throughput is 26.4 MB/sec (<1% utilization). For the native MTU case the QPIP prototype outperforms the other implementations at 75.6 MB/sec with <1% CPU utilization when the other inter-network implementations consume half to ¾ of a host processor. For the smaller MTUs, the limited CPU capacity of the

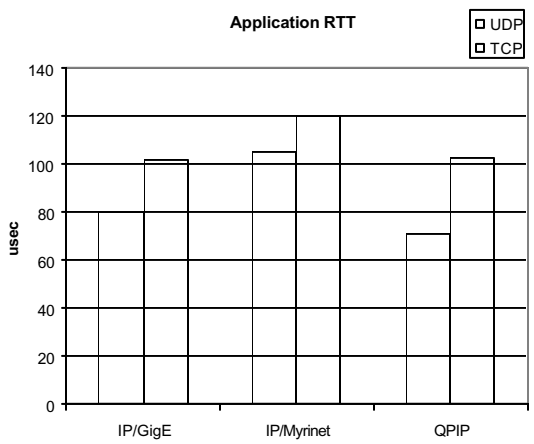

Figure 3. Application-to-Application round trip time.

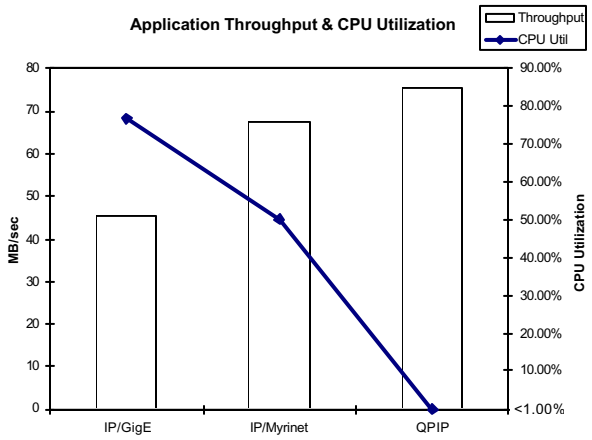

Figure 4. Application-to-Application throughput and CPU utilization.

interface becomes apparent and performs 22% less than the gigabit Ethernet in the 1500 Byte MTU case at 35.4 MB/sec. For the 9000 Byte MTU, QPIP outperforms the IP over Myrinet case at 70.1 MB/sec. In both cases the CPU utilization in the QPIP prototype is <1% thus exhibiting the lightweight nature of the interface.

4.2.2. Overhead analysis. Table 1 presents the send and receive host overhead for a 1 byte message using TCP for the host-based and QPIP implementations The overhead for the host-based inter-network stack was determined by measuring RTT through the loopback interface on an individual host. These results are a lower bound on overhead for a message sent through the host-based network stack because they do not include instructions executed by a particular interface driver. The QPIP overhead measurements were determined by directly timing the associated communication methods from user-space.

Table 1. Host overhead for transmit and receive paths.

	Time (µsec)	Cycles
Host-based IP	29.9	16445
QPIP	2.5	1386

One of the potential bottlenecks of offloading inter-network protocols is how much time is occupied in the network interface for protocol processing. Tables 2 & 3 present an occupancy breakdown for the transmit and receive side TCP processing, respectively, for the QPIP prototype. The measurements were made using the LANai 9 cycle counter and assume a hardware assisted IP checksum on the receive side as described above. There are different processing costs depending on whether the message is data or an acknowledgement (ACK). On the transmit side, an

Table 2. Transmit side network interface processing costs.

Stage	Data Send Time (µsec)	ACK Send Time (µsec)
Doorbell Process	1	1
Schedule	2	2
Get WR	5.5	-
Get Data	4.5	-
Build TCP Hdr	5	5
Build IP Hdr	1	1
Send	1	1
Update	1.5	1.5

Table 3. Receive side network interface processing costs.

Stage	Data Recv Time (µsec)	ACK Recv Time (µsec)
Doorbell Process	1	1
Media Rcv	1	1
IP Parse	1.5	1.5
TCP Parse	7	14
Get WR	5.5	-
Put Data	4.5	-
Update	1.5	9 *(WR and QP State)*

ACK is simply a special case of a data send. On the receive side, an ACK causes RTT estimators and other state to be updated which invokes other processing costs.

For the receive side ACK processing, parsing the TCP header induces a high processing cost because of a series of multiply operations for the RTT estimators. The LANai 9 processor has no hardware multiply, thus these operations are implemented in software. A more specialized interface design would dramatically reduce these costs.

4.2.3. Network storage application. The Network Block Device (NBD) is a client-server application where client block I/O requests are forwarded to a server that emulates a network attached disk (Figure 5). NBD is distributed as part of the Linux kernel and includes a client-side driver and user-level server application. We modified both to use QPIP (Figure 6) and compared client performance to the sockets-based implementation. The benchmark was a 409 MB sequential read and write over an ext2 file system. Writes were flushed to disk using 'sync' and the device was un-mounted between reads to invalidate the client buffer cache. For the QPIP implementation a 9000 Byte MTU was used. Figure 7 presents the results for throughput and CPU efficiency (in MBytes transferred per CPU-second). The QPIP based NBD provides a 40% to 137% throughput performance improvement at up to 133% better CPU effectiveness. Note, for all three implementations, the raw CPU utilization during the benchmark is at least 26% for filesystem processing. For QPIP, none of this is associated with the TCP/IP stack as this is entirely within the adapter.

Integrating the QP interface into NBD was straightforward and proved simpler than the socket implementation by eliminating multiple socket calls and OS specific wrappers to enable kernel-level socket operation.

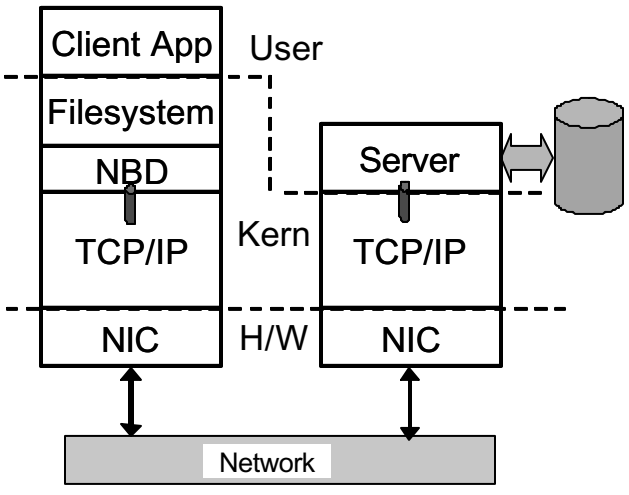

Figure 5: Socket based NBD. Dashed lines represent the user, kernel and hardware boundaries

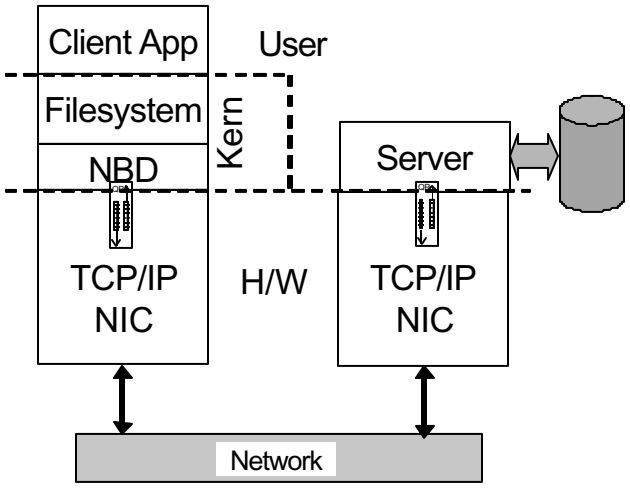

Figure 6: QPIP based NBD.

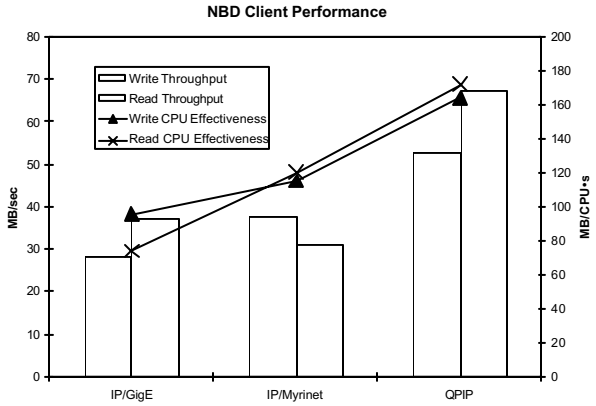

Figure 7. NBD client throughput and CPU effectiveness.

5. Analysis

The QPIP prototype demonstrates the feasibility of achieving a low-overhead interface to IP based networks using the QP abstraction. In this section we present the lessons learned and the implications they have for both inter-network protocols and industry proposed SAN standards such as Infiniband.

5.1. Implications for inter-network protocols

Employing the QP abstraction as the communication interface to inter-network protocols has distinct advantages. First, it provides a virtualized, protected channel from the application to the network that enables tighter integration with the host platform [26]. It also provides the ability to exploit cache coherency mechanisms. For example, applications polling a CQ can spin in the processor cache, as opposed to un-cached accesses across the memory bus. The QP interface directly extends to kernel based applications, as well, without complex software wrappers.

Second, using the QP communication abstraction greatly simplifies the buffer management schemes of the protocol implementation. In traditional systems, the protocol stack in conjunction with the operating system uses mechanisms, such as *mbufs* or *fbufs,* to reduce buffer size and management overhead. The QP model eliminates these complex schemes through straightforward direct data placement. Also, the QP model provides a transparent method of dynamically tuning the receiver window: the more receive buffer space posted, the larger the TCP receive window the sender can utilize. With sockets, adjusting these parameters requires system calls and/or setting OS specific variables.

Third, QPIP also simplifies the mechanisms for scheduling transmit and receive operations. Host based inter-network stacks are dependent on the use of the operating system scheduler. Operating system schedulers must support several types of services which tend to introduce complexity. By placing the inter-network stack beneath the QP in an intelligent interface, the scheduling mechanism can be tailored to meet the needs of the network; effectively an inter-network runtime system. This is indicated in Table 2 by the short scheduling overhead relative to other processing components on the NI.

Finally, QPIP demonstrates that employing an alternative interface to inter-networks enables significant gains for network based I/O. The QP interface provides a simple bridge between common I/O abstractions and network protocols. It is able to take advantage of zero-copy mechanisms and offloaded processing without the constraints of using a sockets interface.

5.2. Implications for System Area Networks

In terms of SAN standards, the QPIP architecture demonstrates a practical means of merging established protocols with a modern SAN usage abstraction to achieve low latency, low overhead and high bandwidth. In our prototype, most of the latency is due to high network interface occupancy as a result of modest hardware support. Even so, it is capable of producing performance comparable if not better than established host based stacks. SAN proposals, such as Infiniband, engineer a high degree of specialized hardware support in the interface. This includes special protocol engines and direct attachment to the host memory subsystem. Functionally, the processing required from native Infiniband protocols would be little different than that in TCP/UDP or IP. Thus, if the same degree of hardware support were to be applied to QPIP then an equivalent performance could be reached. From our prototype results, key areas for hardware support include lightweight doorbell mechanisms, IP checksums, UDP/TCP connection de-multiplexing and advanced mathematical functions.

With respect to Infiniband in particular, QPIP introduces some benefits over the existing specification. First it employs a simple set of packet formats for communication. There is no distinction between packet formats for inter and intra subnet packets. This simplifies protocol processing and enables a straightforward method to bridge the SAN to external networks as described in section 3. It also brings the collection of well-understood transport mechanisms and behaviors to the SAN that do not exist in the present specification. These mechanisms could either be end-to-end or could include network-based mechanisms such as RED or ECN. Inter-network protocols do not bar the use of intelligence in the SAN fabric that can improve performance.

6. Conclusions and Future Work

The integration of the network as a first-class citizen in system architectures presents unique challenges for SAN design. The network must be capable of meeting the demands of low latency, low overhead and high bandwidth. Inter-network architectures, which have been very successful because of their flexibility and open development, are perceived to be unable to meet such demands. Thus, efforts to develop a new standard SAN architecture, such as Infiniband, have focused on completely re-engineering network stacks including the abstraction, transport layer and network layer. The QPIP architecture is based on the premise that standard inter-networks *are* capable of performing in the SAN. By adopting the lightweight queue pair interface of the Infiniband specification and using this as the interface to IP networks, host overhead is significantly reduced while simultaneously enabling efficient offload of the protocol stack. Results from our prototype demonstrate that, with modest hardware, latency and bandwidth equivalent to modern inter-network implementations on powerful processors can be achieved at a fraction of the overhead. Further, our results suggest that applying the degree of hardware support expected for Infiniband to QPIP, similar performance targets of latency and throughput can be achieved. While tailored hardware can deliver performance, we also note that a programmable aspect to the interface is also important for higher level operations. It allows for changes as inter-network protocols continually evolve and introduce new functionality and permits for a degree of flexibility in terms of scheduling and monitoring.

7. Acknowledgements

The authors thank Alan Mainwaring and Kevin Fall for their support and guidance. This work was supported by Intel Corporation, Compaq Corporation, Microsoft Corporation, DARPA contract no. N66001-99-2-8913, LLNL Memorandum Agreement no. B504962 under DOE contract No. W-7405-ENG-48, NSF Infrastructure Grant no. EIA-9802069 and by NASA contract no. NAGII-1210. The information presented here does not necessarily reflect the position of the Government and no official endorsement should be inferred.

8. References

[1] M. B. Abbott and L. L. Peterson, "Increasing network throughput by integrating protocol layers," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 600-10, 1993.
[2] Alacritech Inc., "Alacritech, Inc.," http://www.alacritech.com, 2001.
[3] P. Barford and M. Crovella, "Critical path analysis of TCP transactions," *IEEE/ACM Transactions on Networking*, vol. 9, pp. 238-48, 2001.
[4] T. Blackwell, "Speeding up protocols for small messages," *Proceedings of ACM SIGCOMM '96*,ACM, Stanford, CA, USA, 1996.

[5] T. Braun and C. Diot, "Protocol implementation using integrated layer processing," *Proceedings of ACM SIGCOMM '95*,ACM, Cambridge, MA, USA, 1995.
[6] D. Comer and D. L. Stevens, *Internetworking with TCP/IP*, 2nd ed. Englewood Cliffs, N.J.: Prentice Hall, 1991.
[7] E. C. Cooper, P. A. Steenkiste, R. D. Sansom, and B. D. Zill, "Protocol implementation on the Nectar communication processor," *Proceedings of ACM SIGCOMM '90*,ACM, Philadelphia, PA, USA, 1990.
[8] C. Dalton, G. Watson, D. Banks, C. Calamvokis, A. Edwards, and J. Lumley, "Afterburner (network-independent card for protocols)," *IEEE Network*, vol. 7, pp. 36-43, 1993.
[9] B. S. Davie, "A host-network interface architecture for ATM," *Proceedings of ACM SIGCOMM '91*,ACM, Zurich, Switzerland, 1991.
[10] P. Druschel and L. L. Peterson, "Fbufs: a high-bandwidth cross-domain transfer facility," *Proceedings of 14th ACM Symposium on Operating Systems Principles*,ACM, Ashville, NC, USA, 1993.
[11] P. Druschel, L. L. Peterson, and B. S. Davie, "Experiences with a high-speed network adaptor: A software perspective," *Proceedings of ACM SIGCOMM '94*,ACM, London, UK, 1994.
[12] C. Dubnicki, L. Iftode, E. W. Felten, and L. Kai, "Software support for virtual memory-mapped communication," *Proceedings of IPPS '96.*,IEEE Comput. Soc. Press, Honolulu, HI, USA, 1996.
[13] A. Edwards and S. Muir, "Experiences implementing a high performance TCP in user-space," *Proceedings of ACM SIGCOMM '95*,ACM, Cambridge, MA, USA, 1995.
[14] A. Edwards, G. Watson, J. Lumley, D. Banks, C. Calamvokis, and C. Dalton, "User-space protocols deliver high performance to applications on a low-cost Gb/s LAN," *Proceedings of ACM SIGCOMM '94*,ACM, London, UK, 1994.
[15] Emulex Corporation, "Emulex Corporation," http://www.emulex.com, 2001.
[16] A. Gallatin, J. Chase, and K. Yocum, "Trapeze/IP: TCP/IP at near-gigabit speeds," *Proceedings of the FREENIX Track. 1999 USENIX Annual Technical Conference Proceedings of the FREENIX Track. 1999 USENIX Annual Technical Conference*,USENIX Assoc., Monterey, CA, USA, 1999.
[17] Infiniband Trade Assoc, "Infiniband Architecture Specification, Release 1.0," vol. Vol 1: Infiniband Trade Assoc. (http://www.infinibandta.org), 2000.
[18] iReady Corporation, "iReady Corporation," http://www.iready.com, 2001.
[19] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance," *RFC 1323*, 1992.
[20] J. Kay and J. Pasquale, "The importance of non-data touching processing overheads in TCP/IP," *Proceedings of ACM SIGCOMM '93*,ACM, San Francisco, CA, USA, 1993.
[21] J. Kay and J. Pasquale, "Profiling and reducing processing overheads in TCP/IP," *IEEE/ACM Transactions on Networking*, vol. 4, pp. 817-28, 1996.
[22] J. Kay and J. Pasquale, "A summary of TCP/IP networking software performance for the DECstation 5000," *Proceedings of 1993 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*,ACM, Santa Clara, CA, USA, 1993.
[23] A. M. Mainwaring and D. E. Culler, "Design challenges of virtual networks: fast, general-purpose communication," *Proceedings of Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*,ACM, Atlanta, GA, USA, 1999.
[24] Microsoft Corporation, "Winsock Direct Path," Microsoft Corporation (http://www.microsoft.com), 2000.
[25] D. Mosberger, L. L. Peterson, P. G. Bridges, and S. O'Malley, "Analysis of techniques to improve protocol processing latency," *Proceedings of ACM SIGCOMM '96*,ACM, Stanford, CA, USA, 1996.
[26] S. S. Mukherjee and M. D. Hill, "Making network interfaces less peripheral," *Computer*, vol. 31, pp. 70-6, 1998.
[27] S. Pakin, M. Lauria, and A. Chien, "High performance messaging on workstations: Illinois fast messages (FM) for Myrinet," *Proceedings of SC '95*,ACM, San Diego, CA, USA, 1995.
[28] D. Perkovic and P. J. Keleher, "Responsiveness without interrupts," *Proceedings of the 13th Association for Computer Machinery International Conference on Supercomputing*,ACM, Rhodes, Greece, 1999.
[29] I. Pratt and K. Fraser, "Arsenic: a user-accessible gigabit Ethernet interface," *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications.*,IEEE, Anchorage, AK, USA, 2001.
[30] L. Prylli and B. Tourancheau, "BIP: a new protocol designed for high-performance networking on Myrinet," *Proceedings IPPS/SPDP '98*,Springer-Verlag, Orlando, FL, USA, 1998.
[31] E. Salo and J. Pinkerton, "Scheduled Transfers OS Bypass API," in *Proceedings of the November '98 HIPPI Ad Hoc Working Meeting.* Mountain View, CA: http://www.hippi.org/cSTAPI.html, 1998.
[32] W. R. Stevens and G. R. Wright, *TCP/IP illustrated*. Reading, Mass.: Addison-Wesley Pub. Co., 1994.
[33] T. von Eicken, A. Basu, V. Buch, and W. Vogels, "U-Net: a user-level network interface for parallel and distributed computing," *Proceedings of Fifteenth AC Symposium on Operating Systems Principles*,ACM, Copper Mountain Resort, CO, USA, 1995.
[34] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser, "Active messages: a mechanism for integrated communication and computation," *Proceedings of ISCA '92. 19th Annual International Symposium on Computer Architecture*,ACM, Gold Coast, Qld., Australia, 1992.
[35] G. Watson, D. Banks, C. Calamvokis, C. Dalton, A. Edwards, and J. Lumley, "AAL5 at a gigabit for a kilobuck," *Journal of High Speed Networks*, vol. 3, pp. 127-45, 1994.